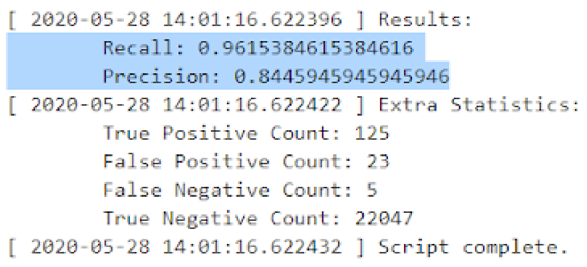
<u>Scoring method: How the product comparison works</u>

Before comparison, the product title and description strings are both preprocessed. This is done by changing a NaN description to an empty string. Afterwards, the stopwords from title and description are removed before stripping it of punctuation. This is so when the strings are compared, only the relevant words are used in the calculation of the similarity score. Secondly, we keep the original product id with their modified title and description by putting it into a dictionary. Afterwards, for each Amazon item, its title and description is compared to every Google item. The comparison is done in question 1a by using fuzzywuzzy's process.extract which picks the best matched string (in google). For title, the comparison function: fuzz.partial_ratio() is used, this is because title is usually shorter than description. Partial is better for partial string matching and returns a high score if the shortest string matches one of the words in the other string. For description, fuzz.token_sort_ratio is used since description could describe things in many different orders and it could get really long. Therefore the function is used as it accounts for strings in different orders by sorting the string first. After both the similarity scores are calculated, the program looks at the highest score and finds the corresponding google id for that amazon item by searching through the dictionary mentioned above.

<u>Evaluation of the overall performance of the product comparison</u>

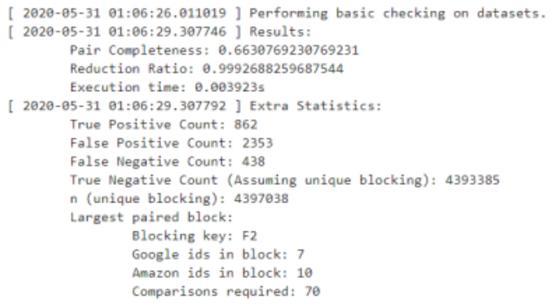


```
[ 2020-05-28 14:01:16.622396 ] Results:
        Recall: 0.9615384615384616
        Precision: 0.8445945945945946
[ 2020-05-28 14:01:16.622422 ] Extra Statistics:
        True Positive Count: 125
        False Positive Count: 23
        False Negative Count: 5
        True Negative Count: 22047
[ 2020-05-28 14:01:16.622432 ] Script complete.
```

The recall is above 0.9, however precision is 0.8. The limitation of the algorithm is that it only matches one amazon item to a google item. The csv of google products are more than the amazon lists. This means that one Amazon item can only be matched with one google item, duplicates are not accounted for. Therefore, if one amazon item has more than one google item to match, it will not do so. To improve, we could loop through the google items and find its best amazon matches and compare which fits better by looking at the scores. However, this increases overall time complexity as comparisons are doubled.

<u>Blocking method: How the blocking method works</u>

Only the title of each item is extracted and is used, because if description was used, the program would have a higher time complexity as it has to compare many strings as the number of items is high. After title extraction, the strings are preprocessed like in task 1a. After this, dictionaries are made where the key is the title of an item and the value is the id. This is to make links between the ids and the new preprocessed titles. Firstly, a similarity function is created using process.extract and partial_ratio is used as it is better for partial string matching, this returns the best matched string (index 0) and its score (index 1) as a list. In the blocking method, the list of google items are used to create the blocks since there are more Google values than Amazon, this is to prevent Google items from being forced into blocks made by Amazon that may not be potential matches. If the blocking dictionary is empty then the first item will be made into a block automatically where it's title is the key and id as the value, if it is not empty then the next item's title will be compared with the blocking key (title), if it is similar enough (score > 85) then the id will be added to the block, if the score < 85 then the title of that item being compared will form a new block in block_dict with a new title as the key. After comparing all google titles to block_dict keys, every Amazon item title will be passed into the similarity function to retrieve the best block (made by google) to belong to. After finding which block it should belong to, the Amazon id is retrieved by using the dictionary made and added to the list of ids in block_dict. After blocking is done, the key of the block_dict (title) is changed to a unique, random combination of 2 characters and outputted to task2b.csv.



```
[ 2020-05-31 01:06:26.011019 ] Performing basic checking on datasets.
[ 2020-05-31 01:06:29.307746 ] Results:
        Pair Completeness: 0.6630769230769231
        Reduction Ratio: 0.9992688259687544
        Execution time: 0.003923s
[ 2020-05-31 01:06:29.307792 ] Extra Statistics:
        True Positive Count: 862
        False Positive Count: 2353
        False Negative Count: 438
        True Negative Count (Assuming unique blocking): 4393385
        n (unique blocking): 4397038
        Largest paired block:
                Blocking key: F2
                Google ids in block: 7
                Amazon ids in block: 10
                Comparisons required: 70
```

<u>Evaluation of the blocking method</u>

In the blocking method, each block is created using google item title as the key. Since Google has more items than Amazon, many items from Amazon might not match into some of the blocks in the dictionary. This causes many blocks to be incomplete. These blocks could be removed as comparisons cannot be made. In addition, for a new block to be created the best match must have a similarity score lower than 85, this might be too high as potential matching items could be added to the wrong block and also it causes the creation of more blocks. This might not be good for my algorithm since each title has to be compared to the titles in block_dict, the more blocks, the more comparisons needed in this blocking method (however less comparisons when actually pairing up the items). So for faster blocking implementation the score threshold could be lower, however sacrificing good pair completeness.